

CLAIMS

1 1. A method for verifying software source code that
2 includes references to program variables, the method
3 comprising:

4 processing the source code to derive a set of
5 next-state functions representing control flow of the
6 source code;

7 replacing the references to the program variables in
8 the source code with non-deterministic choices in the
9 next-state functions;

10 restricting the next-state functions including the
11 non-deterministic choices to produce a finite-state model
12 of the control flow; and

13 verifying the finite-state model to find an error in
14 the source code.

1 2. A method according to claim 1, wherein processing
2 the source code comprises extracting a program counter
3 from the source code, and expressing the next-state
4 functions in terms of the program counter.

1 3. A method according to claim 2, wherein processing
2 the source code further comprises expressing the
3 next-state functions with reference to a stack pointer
4 associated with a stack used in running the code, and
5 wherein replacing the program variables comprises
6 eliminating substantially all the references to the
7 program variables from the next-state functions, leaving
8 the next-state functions dependent on the program counter
9 and on the stack pointer.

1 4. A method according to claim 3, wherein restricting
2 the next-state functions comprises limiting the stack

41260S2

3 pointer to a value no greater than a predetermined
4 maximum.

1 5. A method according to claim 1, wherein replacing the
2 program variables comprises eliminating the references to
3 the program variables from the next-state functions, so
4 that the finite-state model is substantially independent
5 of data values of the program variables.

1 6. A method according to claim 1, wherein processing
2 the source code further comprises expressing the
3 next-state functions with reference to a stack used in
4 running the code, and wherein restricting the next-state
5 functions comprises limiting the stack to a depth no
6 greater than a predetermined maximum.

1 7. A method according to claim 6, wherein expressing
2 the next-state functions comprises expressing the
3 next-state functions in terms of a stack pointer
4 associated with the stack, and wherein limiting the stack
5 comprises limiting the stack pointer to a value no
6 greater than the predetermined maximum.

1 8. A method according to claim 7, wherein expressing
2 the next-state functions in terms of the stack pointer
3 comprises incrementing the stack pointer in response to a
4 function call in the source code, up to the predetermined
5 maximum, and decrementing the stack pointer when the
6 function returns.

1 9. A method according to claim 1, wherein verifying the
2 finite-state model comprises checking the finite-state
3 model against a specification using a model checker.

1 10. A method according to claim 9, wherein restricting
2 the next-state functions comprises automatically

3 producing the model from the source code in a form
4 suitable for processing by the model checker,
5 substantially without human intervention in deriving and
6 restricting the next-state functions or in replacing the
7 references.

1 11. A method according to claim 9, wherein checking the
2 finite state model comprises checking the model against
3 one or more formulas expressed in terms of temporal
4 logic.

1 12. A method according to claim 9, wherein checking the
2 finite state model comprises finding a counter-example
3 indicative of the error.

1 13. Apparatus for verifying software source code that
2 includes references to program variables, the apparatus
3 comprising a program analyzer, which is arranged to
4 process the source code so as to derive a set of
5 next-state functions representing control flow of the
6 source code and to replace the references to the program
7 variables in the source code with non-deterministic
8 choices in the next-state functions, and further to
9 restrict the next-state functions including the
10 non-deterministic choices to produce a finite-state model
11 of the control flow, which can be checked by a model
12 checker to find an error in the source code.

1 14. Apparatus according to claim 13, wherein the program
2 analyzer is arranged to extract a program counter from
3 the source code, and to express the next-state functions
4 in terms of the program counter.

1 15. Apparatus according to claim 14, wherein the program
2 analyzer is further arranged to express the next-state

3 functions with reference to a stack pointer associated
4 with a stack used in running the code, and to eliminate
5 substantially all the references to the program variables
6 from the next-state functions, leaving the next-state
7 functions dependent on the program counter and on the
8 stack pointer.

1 16. Apparatus according to claim 15, wherein the program
2 analyzer is arranged to limit the stack pointer to a
3 value no greater than a predetermined maximum.

1 17. Apparatus according to claim 13, wherein the program
2 analyzer is arranged to remove the references to the
3 program variables from the next-state functions, so that
4 the finite-state model is substantially independent of
5 data values of the program variables.

1 18. Apparatus according to claim 13, wherein the program
2 analyzer is arranged to express the next-state functions
3 with reference to a stack used in running the code, which
4 is limited to a depth no greater than a predetermined
5 maximum.

1 19. Apparatus according to claim 18, wherein the
2 next-state functions are expressed in terms of a stack
3 pointer associated with the stack, and wherein the stack
4 pointer is limited to a value no greater than the
5 predetermined maximum.

1 20. Apparatus according to claim 19, wherein in the
2 next-state functions, the stack pointer is incremented in
3 response to a function call in the source code, up to the
4 predetermined maximum, and is decremented when the
5 function returns.

- 1 21. Apparatus according to claim 13, and comprising a
2 model checker, which is arranged to check the
3 finite-state model against a specification.
- 1 22. Apparatus according to claim 21, wherein the program
2 analyzer is arranged to automatically produce the model
3 from the source code in a form suitable for processing by
4 the model checker, substantially without human
5 intervention in deriving and restricting the next-state
6 functions or in replacing the references.
- 1 23. Apparatus according to claim 21, wherein the model
2 checker is arranged to check the model against one or
3 more formulas expressed in terms of temporal logic.
- 1 24. Apparatus according to claim 21, wherein the model
2 checker is arranged to find a counter-example indicative
3 of the error.
- 1 25. A computer software product for verifying source
2 code that includes references to program variables, the
3 product comprising a computer-readable medium in which
4 program instructions are stored, which instructions, when
5 read by the computer, cause the computer to process the
6 source code so as to derive a set of next-state functions
7 representing control flow of the source code and to
8 replace the references to the program variables in the
9 source code with non-deterministic choices in the
10 next-state functions, and further cause the computer to
11 restrict the next-state functions including the
12 non-deterministic choices to produce a finite-state model
13 of the control flow, which can be checked by a model
14 checker to find an error in the source code.

1 26. A product according to claim 25, wherein the
2 instructions cause the computer to extract a program
3 counter from the source code, and to express the
4 next-state functions in terms of the program counter.

1 27. A product according to claim 26, wherein the
2 instructions cause the computer to express the next-state
3 functions with reference to a stack pointer associated
4 with a stack used in running the code, and to eliminate
5 substantially all the references to the program variables
6 from the next-state functions, leaving the next-state
7 functions dependent on the program counter and on the
8 stack pointer.

1 28. A product according to claim 27, wherein the
2 instructions cause the computer to limit the stack
3 pointer to a value no greater than a predetermined
4 maximum.

1 29. A product according to claim 25, wherein the
2 instructions cause the computer to remove the references
3 to the program variables from the next-state functions,
4 so that the finite-state model is substantially
5 independent of data values of the program variables.

1 30. A product according to claim 25, wherein the
2 instructions cause the computer to express the next-state
3 functions with reference to a stack used in running the
4 code, which is limited to a depth no greater than a
5 predetermined maximum.

1 31. A product according to claim 30, wherein the
2 next-state functions are expressed in terms of a stack
3 pointer associated with the stack, and wherein the stack

4 pointer is limited to a value no greater than the
5 predetermined maximum.

1 32. A product according to claim 31, wherein in the
2 next-state functions, the stack pointer is incremented in
3 response to a function call in the source code, up to the
4 predetermined maximum, and is decremented when the
5 function returns.

1 33. A product according to claim 25, wherein the
2 instructions further cause the computer to check the
3 finite-state model against a specification.

1 34. A product according to claim 33, wherein the
2 instructions cause the computer to automatically produce
3 the model from the source code in a form suitable for
4 checking against the specification, substantially without
5 human intervention in deriving and restricting the
6 next-state functions or in replacing the references.

1 35. A product according to claim 33, wherein the
2 instructions cause the computer to check the model
3 against one or more formulas expressed in terms of
4 temporal logic.

1 36. A product according to claim 33, wherein the
2 instructions cause the computer to find a counter-example
3 indicative of the error.